VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KOSTERNÍ ANIMACE PRO GPUENGINE

SKELETAL ANIMATION FOR GPUENGINE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. ANTONÍN MINAŘÍK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. TOMÁŠ STARKA**
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Minařík Antonín, Bc.**
Program: Informační technologie Obor: Počítačová grafika a multimédia
Název: **Kosterní animace pro GPUengine**
Skeletal Animation for GPUengine
Kategorie: Počítačová grafika

Zadání:

1. Nastudujte OpenGL, GPUEngine a teorii potřebnou k animacím.
2. Implementujte část pro načtení animací ze souboru.
3. Implementujte rozšíření knihovny GPUEngine pro kosterní animace.
4. Implementujte demonstrační aplikaci.
5. Vytvořte plakát nebo video k práci.

Literatura:

- Po dohodě s vedoucím

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Starka Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 6. listopadu 2018

Abstrakt

Cílem této práce je studium technik používaných pro kosterní animace a následný návrh a implementace rozšíření pro kosterní animace do knihovny GPUEngine. V teoretické části jsou popsány techniky animací, kosterních animací a skinningu. Následuje rozbor existujících systémů kosterních animací. Navržené řešení se snaží o nízkou redundanci dat v paměti při vykreslování více kosterních modelů. Podle návrhu byl implementován základní systém kosterních animací. Dále byla vytvořena demonstrační aplikace ukazující jeho použití. Výsledný kosterní systém lze použít v jednoduchých 3D aplikacích a může sloužit jako základ pro další práce.

Abstract

This paper deals with studying skeletal animation techniques, and the subsequent design and implementation of skeletal animation extension for the GPUEngine library. The theoretical part describes the techniques of animation, skeletal animation and skinning. The following is an analysis of existing skeletal animation systems. The proposed solution seeks to reduce the data redundancy in the memory while rendering more skeletal models. According to the design a basic skeletal animation system has been implemented. Furthermore, a demonstration application has been created showing the skeletal system's use. The resulting skeletal system can be used in simple 3D applications and can serve as a basis for further works.

Klíčová slova

animace, kosterní animace, keyframing, vertex blending, inverzní kinematika, skinning, linear blend skinning, multi-weight enveloping, animation space, dual quaternion skinning, assimp, OGRE, OpenSceneGraph, GPUEngine

Keywords

animation, skeletal animation, keyframing, vertex blending, inverse kinematics, skinning, linear blend skinning, multi-weight enveloping, animation space, dual quaternion skinning, assimp, OGRE, OpenSceneGraph, GPUEngine

Citace

MINAŘÍK, Antonín. *Kosterní animace pro GPUengine*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Starka

Kosterní animace pro GPUengine

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana inženýra Tomáše Starky. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Antonín Minařík
22. května 2019

Obsah

1 Úvod **3**

2 Teorie **4**
2.1 Počítačová animace 4
2.1.1 Keyframing 4
2.1.2 Per-vertex animace 5
2.1.3 Morph targets 5
2.2 Historie kosterních animací 5
2.3 Principy kosterních animací 6
2.3.1 Vertex blending 8
2.3.2 Animation retargeting 9
2.3.3 Míchání kosterních animací 9
2.3.4 Motion capture 10
2.3.5 Inverzní kinematika 11
2.3.6 Rag doll 11
2.4 Skinning 11
2.4.1 Linear blend skinning 12
2.4.2 Multi-linearní metody 14
2.4.3 Nelineární metody 14
2.4.4 Transformace normálových vektorů 16
2.5 Existující řešení systémů kosterních animací 16
2.5.1 OGRE 17
2.5.2 OpenSceneGraph 18

3 Návrh rozšíření **20**
3.1 GPUEngine 20
3.2 Analýza 22
3.3 Návrh a požadavky 22
3.4 Načtení modelu s animacemi ze souboru 22

4 Implementace **26**
4.1 Systém kosterních animací 26
4.1.1 Řešené implementační problémy 26
4.1.2 Detaily Implementace 27
4.2 Načtení modelu s animacemi ze souboru 29
4.3 Použití v aplikaci 30
4.4 Demonstrační aplikace 31

5 Možnosti budoucí práce **33**

6 Závěr **34**

Literatura **35**

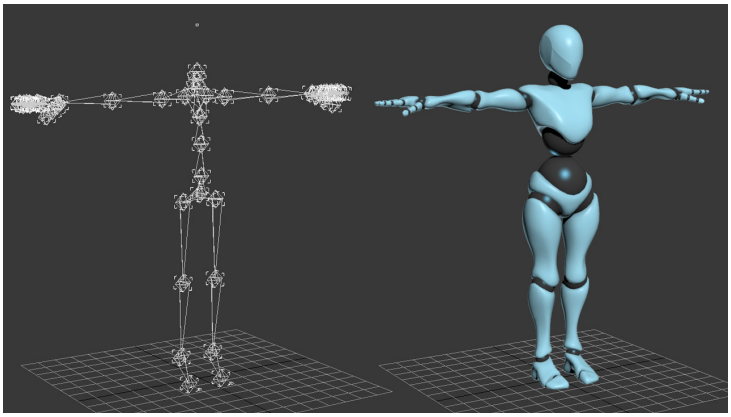
Kapitola 1

Úvod

Animace slouží k vytvoření iluze pohybu pomocí zobrazování posloupnosti obrázků. Tyto obrázky se však někde musejí vzít. Ruční tvorba všech snímků animace je zdlouhavá, a proto existují způsoby pro usnadnění práce. Ve 2D grafice se obvykle scéna skládá z menších obrázků (sprajtů), kterými lze samostatně pohybovat a animovat je. Animace ve 3D grafice jsou mnohem složitější. Objekty jsou zde reprezentovány modely složenými z trojúhelníků. Klasický způsob tvorby animace takového modelu zahrnuje určení pozic všech jeho trojúhelníků, což je časově i paměťově náročné. Kosterní animace zavádí kostru skládající se z kostí. Tato kostra se chová podobným způsobem jako ta lidská (například při pohybu paží se příslušným způsobem pohne i ruka s prsty). Při tvorbě kosterních animací stačí nastavit kostru, což práci hodně zjednoduší. Deformace modelu je automaticky řízena podle nastavení kostí kostry. Obrázek 1.1 ukazuje možnou kostru a její model.

Knihovna GPUEngine poskytuje nástroje usnadňující programování 3D grafických aplikací. Nepodporuje však kosterní animace. Tato práce se zabývá implementací systému kosterních animací do této knihovny a následnou demonstrací jeho použití v ukázkové aplikaci.

V první části jsou popsány techniky využívané pro animace a hlavně kosterní animace. Následuje rozbor technik pro skinning, což je nejsložitější část kosterních animací. Nakonec je popsán stav implementace rozšíření pro GPUEngine a plánované pokračování.



Obrázek 1.1: Kostra s modele v tzv. T-póze.[1]

[1] http://www.downloads.redway3d.com/downloads/public/documentation/wf_skeletal_animation.html

Kapitola 2

Teorie

Technika kosterních animací je podmnožina počítačové animace, proto jsou v této kapitole nejprve popsány obecné principy využívané v počítačové animaci. Následuje část o historii a prvním použití kosterních animací. A poté principy samotných kosterních animací. Dále následuje popis skinningu, který slouží pro konečné vykreslení animovaného modelu. Nakonec je zařazena sekce o existujících řešeních systémů kosterních animací v nastudovaném softwaru.

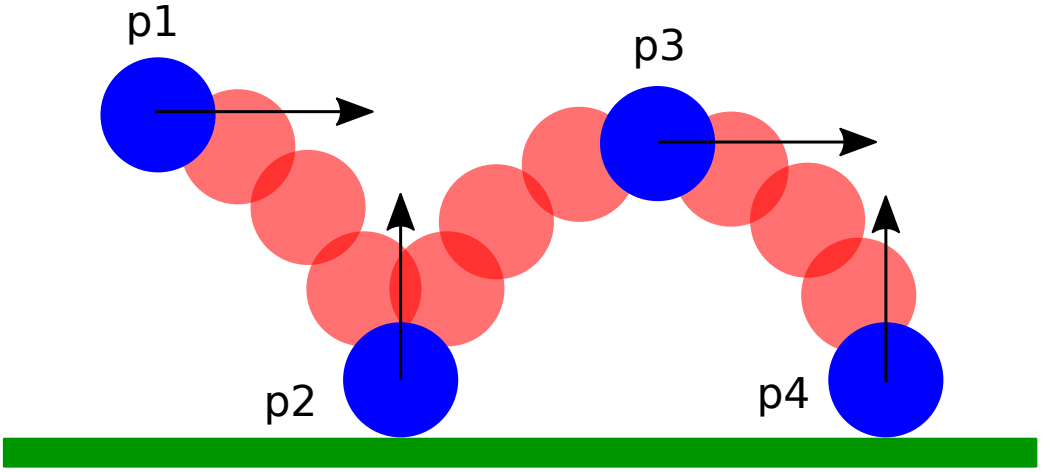
2.1 Počítačová animace

Při sledování posloupnosti na sebe navazujících obrázků v rychlé sledu splynou dohromady a může vzniknout dojem pohybujících se objektů. Jednotlivým obrázkům této iluze se říká snímky (frames) a celé sekvenci animace. Nemusí při tom být využito počítače jako například při promítání analogového filmu, tato práce se však zaměřuje pouze na počítačovou animaci. Počítačová animace by se dala definovat jako technika, při které je iluze pohybu tvořena zobrazováním na obrazovku (nebo zaznamenáváním) posloupnosti jednotlivých stavů dynamické scény [19].

Jako první je v této kapitole popis techniky keyframe animací, která se v nějaké formě používá téměř všude, kde se používají animace (včetně kosterních animací). Následuje část popisující per-vertex animace, které tvoří ten nejzákladnější způsob animací. Následuje popis morph targets, využívající extrémy pro vyjádření stavů mezi nimi.

2.1.1 Keyframing

Keyframe animace (animace podle klíčových snímků) jsou založeny na automatickém generování mezisnímků nazývaných in-betweens založených na sadě klíčových snímků dodaných animátorem. Existují dvě různé metody keyframe animací – per-vertex a parametrická. V per-vertex jsou in-betweens založeny na interpolaci bodů obrazu či vrcholů modelu. U parametrických keyframe animací je zobrazovaný objekt popsán parametry (např.: konfigurací kostry). Animátor vytvoří klíčové snímky specifikací sady parametrů v daném čase, parametry jsou interpolovány a výsledné obrazy jsou poté jednotlivě vytvořeny z interpolovaných parametrů. Lineární interpolace vytváří nespojitosti na úrovni první derivace, což vede k nerovnoměrné rychlosti, a tím pádem trhavým animacím. Proto je obvykle pro lepší výsledky nutné použít kubickou interpolaci nebo interpolaci na základě splinů (např. Kochanek-Bartels spline [7]). Na obrázku 2.1 je vidět princip keyframe animací. [19]



Obrázek 2.1: Keyframing je základní technika používaná v počítačových animacích. V kosterních animacích je této techniky také hojně využíváno. Červené mezisnímky jsou automaticky vypočítány podle zadaných modrých klíčových snímků. Interpolované parametry jsou pozice míčku v klíčových snímcích a vektory jeho pohybu.

2.1.2 Per-vertex animace

Per-vertex animace (animace po vrcholech) je ten nejzákladnější způsob animací. Dovoluje největší volnost, protože umožňuje pohybovat s každým vrcholem modelu samostatně. Je však extrémně datově náročný, pro každý vrchol modelu pro každý klíčový snímek je třeba uložit buď jeho pozici nebo změnu od té minulé. Kvůli tomuto nedostatku se při renderování v reálném čase obvykle nepoužívá. [3]

Dalo by se říci, že každá animace se při vykreslování stává per-vertex animací, protože na obrazovce jsou vidět výsledné vrcholy modelu.

2.1.3 Morph targets

Morph targets jsou podobné per-vertex animacím, protože ukládají data, pro všechny vrcholy modelu. Rozdíl je však v tom, že těchto verzí modelu ukládají pouze omezené množství. Animátor vytvoří sadu verzí modelu tak, že má kontrolu nad každým jednotlivým vrcholem. Mezi nimi je poté lineárně interpolováno a tak vznikají animace. Tato technika se velmi často používá pro animace tváří, protože lidská tvář je nesmírně komplexní a ovládá ji přibližně 50 svalů. Na obrázku 2.2[1] je vidět využití této techniky. [3]

2.2 Historie kosterních animací

S technikou kosterních animací přišli Nadia Magnenat Thalmann, Richard Laperrière a Daniel Thalmann a v roce 1988 publikovali v článku [11]. Techniku využili pro animování postav v počítačově animovaném filmu Rendez-vous in Montreal z roku 1987 [12]. Jejich kostra obsahuje články a klouby, mezi články může být úhel nazývaný úhel kloubu. Pohyb je určen nastavením úhlů kloubu v klíčových snímcích a mezihodnoty jsou interpolovány pomocí bikubických splinů [7]. Již v této technice plně oddělovali topologii povrchu od

[1] http://developer.download.nvidia.com/books/HTML/gpugems/gpugems_ch04.html

Obrázek 2.2: Morph targets použité pro animaci výrazů v obličeji. Nastavením různé váhy lze plynule přecházet mezi výrazy či vyjádřit míru nějaké emoce postavy.

kostry, aby bylo možné využít různé metody pro určení povrchu modelu. Na obrázku 2.3 je vidět scéna z filmu, při které postava zvedá skleničku ze stolu.

Od prvního využití se kosterní animace prosadily a jsou hojně využívány především ve filmovém a videoherním průmyslu. V dnešní době jde o standardní způsob animování postav i jiných modelů, na které jde napasovat kostra. Podle požadavků mohou být model postavy i jeho animace různě složité. Sofistikované postavy mohou zahrnovat různě složité rozčlenění včetně animací obličeje.

2.3 Principy kosterních animací

Tvorba animací pro trojrozměrný model obsahující stovky či tisíce vrcholů a polygonů je náročná. Tato úloha může být do velké míry zjednodušena seskupením části vrcholů do skupin tvořících části těla modelu, které se pohybují jednotně (přísluší jedné kosti). Ty jsou následně spojeny sadou kloubů. Například lidský model může být rozdělen na části spojené krkem, rameny, lokty, zápěstími, boky, koleny a kotníky. Seskupení částí modelu a určení kloubů může být závislé na složitosti animace. Pro jednoduché modelování chůze lze například ruce a nohy považovat za jediné samostatné díly, které se pohybují relativně k hlavní části těla. Identifikaci částí modelu a přidělování kostí se říká rigging a zabývá se jím modelář či animátor. [16]

Kostra je hierarchická struktura tvořená z kostí, jež jsou definovány posunutím od rodičovské kosti a rotací. Na obrázku 2.4 jsou vidět modely s kostrou použitelnou pro jejich animování.

Určitému nastavení kostí kostry se říká póza. Je určena úhly, které mezi sebou jednotlivé kosti svírají. Při tvorbě modelu je důležitá jedna specifická póza tzv. bind pose (výchozí póza, doslovně přeloženo svazující, připojující). Tato póza spojuje model a kostru. Trojú-

Obrázek 2.3: Marilyn Monroe ve filmu Rendez-vous in Montreal. Ruka zvedající skleničku je řízena kosterními animacemi. [12]

helníky modelu ji vždy zaujímají a kostra ve výchozím stavu také, jinak by transformace neprobíhaly správně. Existuje několik výchozích póz, které se využívají, především T-póza a A-póza. Pojmenovány jsou podle písmene, které póza připomíná. Obě mají výhody i nevýhody, záleží na využití daného modelu. T-póza má jednodušší tvar (kosti v ní svírají pravé úhly). A-póza je přirozenější (horní končetiny jsou přibližně v polovině svého úhlového rozsahu). Z hlediska programátora v tom není rozdíl, důležité je, aby vstupní model a kostra k sobě pasovaly. Na obrázku 2.4 lze vidět T-pózu a A-pózu.

Kosterní animace dovolují nastavit pózu modelu pomocí pouze několika málo parametrů (transformací kostí). Takovéto snížení parametrů vede ke snížení jak paměťové náročnosti, tak náročnosti vytváření animace.

Existují dva způsoby jak může být model tvořen. Poskládán z několika částí, kde každá je nezávisle transformována a přesunuta na příslušnou pozici a modely tvořené jedním meshem (jedinou sítí trojúhelníků), jejichž skupiny vrcholů jsou různě transformovány.

V případě skládaného modelu je každá komponenta nejprve vytvořena ve vlastním lokálním systému souřadnic. Poté je aplikována posloupnost transformací na základě toho, kde se část nachází v rámci modelu. Princip je zde podobný, jako u grafu scény a takhle definovaný model by se v běžném grafu scény dal postavit.

Model určený jediným meshem vyžaduje poměrně odlišný přístup. Dochází k jinému způsobu transformování souřadnic, protože všechny vrcholy meshe jsou určeny ve společném referenčním souřadnicovém systému. Nicméně v případě stejné topologie kostry obou modelů tvořených těmito rozdílnými způsoby je možné aplikovat stejně definovanou animaci a dostat se k podobnému pohybu. Je to možné díky tomu, že pro model tvořený jediným meshem je vytvořena virtuální kostra a vrcholy jsou přiděleny jednotlivým kostem (s případnými váhami přidělení). Transformace jsou potom prováděny na kostře a podle ní jsou posléze transformovány vrcholy modelu. Transformování vrcholů modelu podle nastavení kostí se nazývá skinning a způsoby jakými je prováděn jsou rozebrány v sekci 2.4.

Obrázek 2.4: Modely ve výchozí póze se zvýrazněnou kostrou. Na obrázku vlevo je červeně zvýrazněna kostra.[4]Na pravém obrázku jsou světle modře kosti.[5]Červená, zelená a modrá vyjadřují osy počátků jejich souřadnicových systémů.

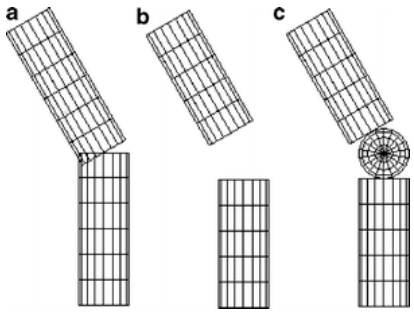
Většinou se používají modely z jednoho meshe, protože dovolují realističtější animace postav. Modely složené z více meshů obvykle trpí jistými problémy, které lze řešit technikou vertex blending, ta je popsána v další části. Následuje popis techniky animation retargeting, vysvětlující možnosti znovupoužití animací. Poté míchání kosterních animací, které je důležité pro jejich plynulost a rozmanitost. Nakonec je zařazeno několik možností tvorby animací, mezi které patří: technika motion capture, která se hojně využívá při vytváření kosterních animací, inverzní kinematika, která animace vytváří pomocí specifického výpočtů a rag doll, využívající fyzikální simulace.

2.3.1 Vertex blending

U poskládaného modelu často dochází k problému při určité orientaci kostí. Když jsou dva meshe otočeny okolo stejného bodu o různé úhly, určité jejich části mohou do sebe proniknout a na druhé straně se zase může objevit mezera. Opravování těchto průniků není jednoduchou úlohou. Proto se pohyblivé části obvykle oddělují malou mezerou, aby se v dovoleném rozsahu pohybu či otočení nedotkly. Někdy se do mezery vkládá koule, aby ji zaplnila, ale to pro většinu modelů není vhodné. Obrázek 2.5 ilustruje tuto problematiku.

Vhodnějším řešením je využití interpolace pro doplnění mezioblasti pohyblivých částí. Proces vytváření povrchu patřícího do této mezioblasti se nazývá vertex blending. Příslušné páry bodů dvou pohyblivých částí jsou propojeny polygony, které dále mohou být děleny

[3]Obrázek převzat z https://www.youtube.com/watch?v=bSRuSW50_4Y
[4]Obrázek převzat z https://developer.valvesoftware.com/wiki/Skeletal_animation.



Obrázek 2.5: (a) Může dojít k nevyžádanému překrytí částí a vzniků mezer. (b) Překrytí lze zabránit vložením mezery, přičemž však problém nespojitosti zůstává. (c) Pro některé modely je možné použít vložení koule na místo kloubu (například pro model robota). [16]

a jejich pozice interpolovány, aby vznikl povrch požadovaného tvaru. Interpolovat lze lineárně, ale mnohem lepší je využít polynomy nebo spliny.

2.3.2 Animation retargeting

Kosterní animace je vždy definována pro jednu kostru. Dává tedy smysl, že je možné použít ji i pro jiný model, který však používá identickou kostru. Animation retargeting dovoluje znovupoužití animací i pro jiné příbuzné kostry. Například pokud jsou dvě kostry identické až na několik listových kostí, je možné animace pro přebývající kosti ignorovat a použít danou animaci i na jednodušší model.

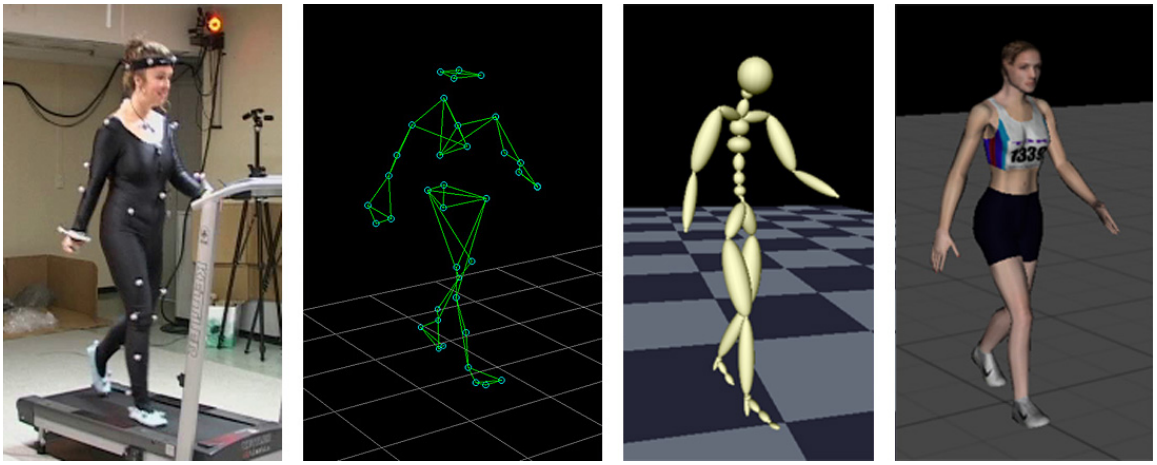
Existují i pokročilejší techniky pro sdílení animací mezi odlišnými kostrami. Jejich výsledky však obvykle nejsou příliš spolehlivé a může být zapotřebí ruční doladění animací.[3]

2.3.3 Míchání kosterních animací

Míchání kosterních animací (animation blending) je technika pro kombinaci více animačních póz dané kostry. Lze ji využít pro mnoho účelů. Nejzákladnější použití míchání kosterních animací je při přehrávání animace. Animační sekvence se obvykle skládá z klíčových snímků (popsáno v 2.1.1), časy pro které je póza vyžadována však obvykle neodpovídají časům jednotlivých snímků. Proto se provádí míchání mezi těmi nejbližšími danému času. Mezi časté využití také patří míchání dvou synchronizovaných animačních sekvencí. Například kombinací chůze a běhu lze vytvořit animace pro různé rychlosti pohybu postavy. Další možností využití je prolínání mezi koncem jedné animace a začátkem druhé, což zajišťuje lepší plynulost přechodu. Dále může být využito znalosti dvou statických póz a míchání použít k nalezení specifické pózy mezi nimi. Například pokud jsou známy pózy postavy ukazující do různých směrů, smícháním lze vytvořit postavu ukazující do libovolného směru mezi nimi.

Míchání více animací v praxi vytváří animace nové. Díky možnosti různého nastavení vah lze ze dvou vstupních animačních sekvencí vytvořit i několik velice odlišných animací.

Pro míchání kosterních animací se využívá lineární interpolace mezi jednotlivými kostmi. Matice nejsou pro lineární interpolaci vhodné. Proto se místo nich používají transformace SQT (scale, quaternion, translation), které obsahují tři složky: scale, rotaci a posun. Scale a posun jsou reprezentovány vektory o třech složkách a rotace kvaternionem. Interpolace



Obrázek 2.6: Na obrázku lze vidět různé fáze techniky motion capture. Nejprve musí herec podat výkon. Poté je identifikována pozice bodů na jeho těle ve virtuálním 3D prostoru. Dále jsou tyto body převedeny na nastavení kostí a ta je nakonec přiřazena výslednému modelu. [7]

vektorů je obyčejnou lineární interpolací mezi jejich složkami. Pro rotaci je použita sférická lineární interpolace zajišťující správnou interpolaci mezi kvaterniony.

Interpolace se provádějí v lokálních prostorech kostí. Při interpolaci v prostoru modelu výsledky vypadají nepřirozeně. Tato skutečnost umožňuje počítat mezi pózy paralelně a je to tak vhodná operace pro urychlení na grafické kartě.

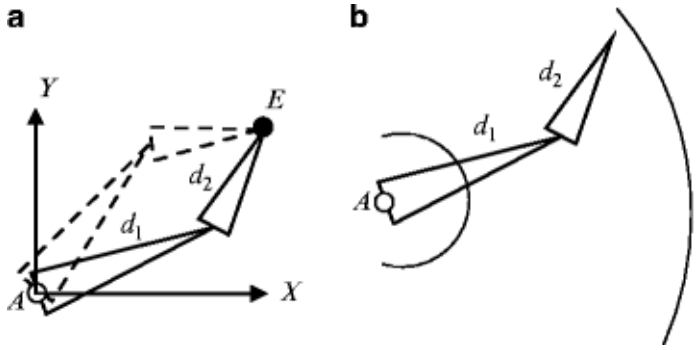
Pokud jsou animační sekvence vytvořeny speciálním způsobem, existuje i možnost jak mezi nimi plynule přecházet bez jejich míchání. Tento způsob se nazývá core poses (základní pózy). Animátor si určí několik základních póz a ví, že jedna animace musí končit a druhá začínat v té samé základní póze. [3]

Additive blending

Additive blending také zajišťuje míchání animací, přistupuje k tomu však odlišným způsobem. Namísto klasické animační sekvence nastavující kosti do určité polohy, jsou použity rozdílové animační sekvence či rozdílové pózy. Ty berou již nastavenou kost a transformaci k ní přidávají. Tato metoda rozšiřuje míchání animací o další řadu možností. Umožňuje kombinovat animace různých částí těla nezávisle na sobě. Například animace úderu postavy, ať už je zbytek jejího těla nastaven jakkoli. Použití rozdílové pózy může zase vést na modifikaci postoje postavy, aniž by bylo třeba měnit ostatní animace. [3]

2.3.4 Motion capture

Motion capture spočívá v nahrání pohybu pro každý snímek nějakým zařízením a využití této informace o pohybu jako základ pro animaci. Například může být nahrána chůze člověka a přenesena na kostru modelu postavy. Oproti ručně vytvářeným kosterním animacím jsou výsledné pohyby mnohem přirozenější. Data z motion capture je však obvykle stejně potřeba doladit zásahem člověka. Zpracování dat z motion capture je většinou jednodušší, než vytváření animací od začátku ručně. Nevýhodou této metody je malá znovupoužitelnost konkrétních pohybů. Na obrázku 2.6 jsou vidět fáze techniky motion capture. [19]



Obrázek 2.7: (a) Nastavení kostí může mít víc řešení. (b) Části kružnic značí místa, pro které existuje pouze jedno řešení a za nimi již neexistuje žádné. [16]

2.3.5 Inverzní kinematika

Inverzní kinematika řeší problém výpočtu pózy modelu či robota tak, aby zakončení splnilo nějaký předurčený cíl. Transformace kostí jsou určeny automaticky. To je klíčové, protože nezávislými proměnnými jsou právě tyto transformace. Jednoduché analytické řešení existuje pouze pro 2 kosti a i to může mít několik řešení. Obrázek 2.7 demonstruje problematiku dvou kostí v dvourozměrném prostoru. Spojení více kostí vede na mnoho možných konfigurací se stejným koncovým bodem. Standardně se inverzní kinematika řeší numerickými metodami jako transpozicí Jacobiho matice [1]. [2]

2.3.6 Rag doll

Rag doll (v překladu hadrová panenka) je způsob procedurální animace založené na fyzikální simulaci tuhých těles. Často se používá ve videohrách pro reprezentaci postav v bezvědomí a postav po smrti. Model se stane bezvládným a chová se jako „hadrová panenka“. Rag doll je kolekce fyzikálně simulovaných tuhých těles, kde každé reprezentuje tuhou část modelu postavy. Tyto části jsou propojeny a omezovány klouby. [3]

2.4 Skinning

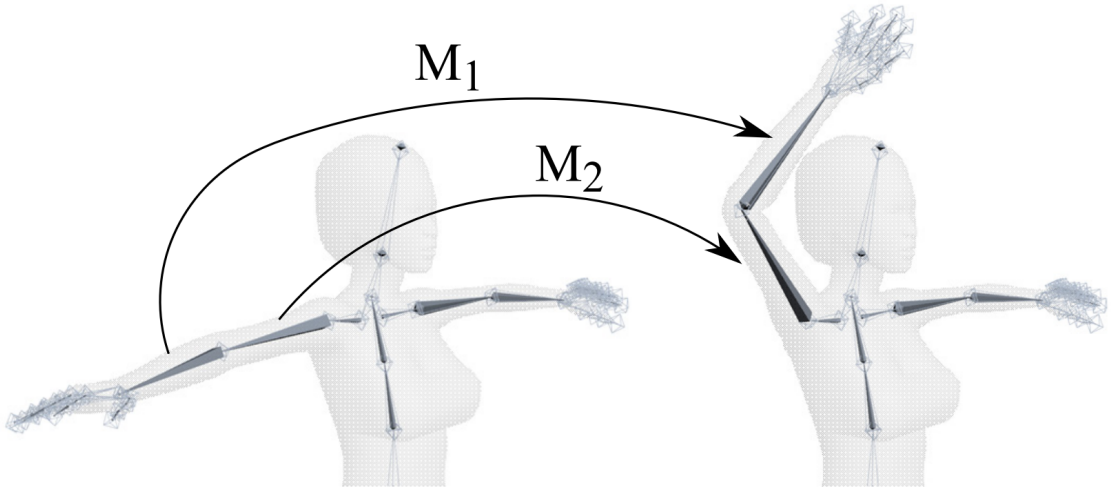
I když je pro animaci modelu využito kostry, ve výsledku se zobrazuje pouze povrch modelu. Termín skinning vychází z anglického slova skin (kůže, slupka) a jde o proces vytváření tohoto povrchu modelu. Vstupními parametry skinningu bývá kostra v nějaké konfiguraci a povrch modelu ve výchozí pozici. Obecně nemusí být skinning součástí pouze kosterních animací, ale můžeme ho chápat jako jakoukoli deformaci v reálném čase, která je určená malou sadou intuitivních parametrů. [4]

Tato podkapitola vysvětluje základní používané techniky skinningu a jejich specifika.

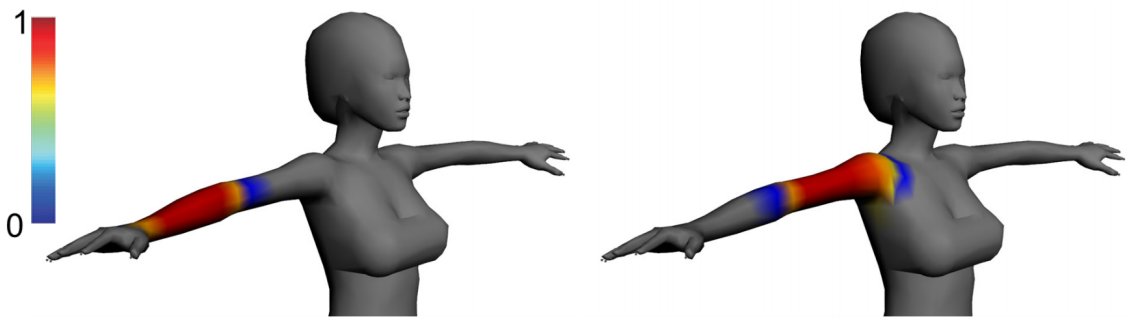
Techniky skinningu lze rozdělit na variační a přímé. Variační metody úlohu pojímají jako optimalizační problém, numericky minimalizující nějakou formu elastické energie. Přímé metody dovolují přímo spočítat výsledný povrch.

Tato část popisuje linear blend skinning, což je základní a nejpoužívanější metoda. Následují multi-lineární metody, jejichž cílem je odstranit některé nedostatky linear blend

[7]Obrázek převzat z http://cta-diderot.brucity.be/?page_id=11



Obrázek 2.8: Ukázka transformace kostí z výchozí pózy pomocí dvou transformačních matic příslušejících kostem.[4]

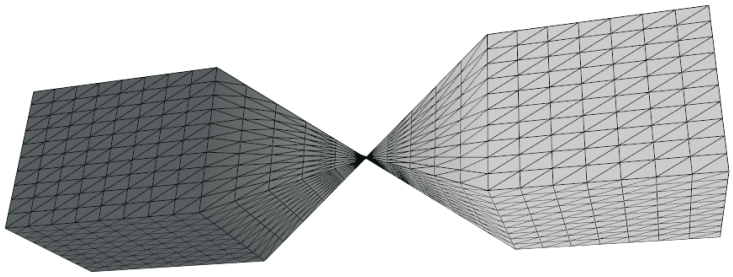


Obrázek 2.9: Tento obrázek demonstruje možné nastavení vah vrcholů pro kost předloktí (vlevo) a kost pažní (vpravo). Váhy vrcholů uprostřed jsou nastaveny na hodnotu 1, protože jejich poloha závisí plně na příslušné kosti. U kraje kostí se jejich vliv na vrchol prolíná.[4]

skinningu přidáním více parametrů. Nakonec jsou uvedeny nelineární metody, které nabízejí ještě lepší výsledky.

2.4.1 Linear blend skinning

Linear blend skinning je základní a nejznámější algoritmus pro přímou deformaci povrchu podle kostry. Přesně matematicky byl poprvé popsán v [10]. Pro svou funkci vyžaduje vstupní data ve formě: modelu ve výchozí póze, transformace kostí a váhy pro skinning. Model ve výchozí póze je obvykle reprezentován polygonovým meshem. Předpokládá se konstantní polygonové propojení modelu, tzn. pouze pozice vrcholů jsou během deformace transformovány. Vrcholy výchozí pózy jsou značeny $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathbb{R}^3$. Často je vhodné rozšířit vrcholy o další rozměr s jedničkovou hodnotou souřadnice pro pohodlné použití s homogenním systémem souřadnic.



Obrázek 2.10: Candy-wrapper artefakt vzniká pokud je transformace navazující kosti rotace o 180 stupňů a je použito linear blend skinningu. [5]

Transformace kostí jsou reprezentovány seznamem matic $\mathbf{T}_1, \ldots, \mathbf{T}_m \in \mathbb{R}^{3\times4}$. Tyto matice odpovídají kostem a transformují výchozí pózu do požadované animované pozice jako třeba na obrázku 2.8. Během animace jsou tyto matice jediné co se mění.

Pro vrchol $\mathbf{v}_i$ máme váhy $w_{i,1}, \ldots, w_{i,m} \in \mathbb{R}$. Každá váha $w_{i,j}$ určuje kolik vlivu má kost j na vrchol i. Obrázek 2.9 ilustruje možné nastavení těchto vah. Častým požadavkem na váhy je nezápornost a jejich součet dávající jedna, tedy $(w_{i,j} \geq 0) \wedge (w_{i,1} + \cdots + w_{i,m} = 1)$.

Pomocí linear blend skinningu se transformovaný vrchol vypočíta podle rovnice:

$$\mathbf{v}'_i = \sum_{j=1}^{m} w_{i,j}\mathbf{T}_j\mathbf{v}_i = \left(\sum_{j=1}^{m} w_{i,j}\mathbf{T}_j\right)\mathbf{v}_i \tag{2.1}$$

Forma nejpravější části rovnice zvýrazňuje skutečnost, že vrchol výchozí pózy $\mathbf{v}_i$ je transformován lineární kombinací transformací kostních matic $\mathbf{T}_j$. Podle této lineární kombinace je odvozen název, transformace jsou lineárně smíchány (anglicky blend). Tyto matice jsou deformačními primitivy (základní stavební bloky) linear blend skinningu. I když mohou být použity libovolné afinní transformace, občas je vhodné předpokládat, že $\mathbf{T}_j$ jsou transformace tuhého tělesa.

Linear blend skinning funguje dobře pokud smíchané transformace nejsou příliš odlišné. Problém nastává, když se snažíme zkombinovat transformace značně se lišící v rotaci. Nejlépe ho lze demonstrovat na těchto dvou transformacích:

$$\mathbf{T}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \mathbf{T}_2 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

$\mathbf{T}_1$ je matice identity, která ponechá vrcholy na stejném místě, $\mathbf{T}_2$ je rotace o 180 stupňů. Kombinace těchto transformací $0.5 \cdot \mathbf{R}_1 + 0.5 \cdot \mathbf{R}_2$ vede na matici hodnosti 1, a tím pádem jsou vrcholy transformovány pouze do jednorozměrného prostoru. Tomuto artefaktu se říká candy-wrapper, protože připomíná způsob zabalení některých bonbonů a jeho ilustraci lze vidět na obrázku 2.10. Linear blend skinning je použitelná základní metoda, pokud není vždy vyžadováno korektní chování nebo je brán ohled na její nedostatky. Další přímé metody berou základní princip této metody a přidávají více parametrů nebo transformací, které zlepšují chování výsledného povrchu. [4]

2.4.2 Multi-linearní metody

Multi-lineární metody berou linear blend skinning a zobecňují ho. Aby předešly candy-wrapper artefaktu, přidávají více váhovacích parametrů pro každé spojení vrchol-kost.

Nejobecnější způsob jak nahlížet na lineární skinning je pomocí této rovnice:

$$\mathbf{v}' = \mathbf{X}\mathbf{t} \tag{2.3}$$

- $\mathbf{X} \in \mathbb{R}^{3n \times 12m}$ je matice parametrů, která je během animace konstantní. Zahrnuje výchozí pózu, souřadnice vektorů a váhy.
- $\mathbf{t} \in \mathbb{R}^{12m}$ zaštiťuje všechny transformace, tj. nastavení kostí. Je to jediný během animace se měnící vstup. Pro zjednodušení zápisu je zde provedena vektorizace, původní matice kostí jsou převedeny na vektory a zřetězeny za sebe, a tím vytvoří vektor $\mathbf{t}$.
- $\mathbf{v}' \in \mathbb{R}^{3n}$ je výsledný transformovaný povrch modelu ve formě pozic vrcholů. [4]

Pro lepší pochopení problematiky je vhodné rozdělit matici $\mathbf{X}$ na bloky 3×12. Tento jeden blok představuje pár vrcholu a kosti. Pro linear blend skinning má tuto podobu:

$$\begin{bmatrix} w_{i,j}v_{i,1} & 0 & 0 & w_{i,j}v_{i,2} & 0 & 0 & w_{i,j}v_{i,3} & 0 & 0 & w_{i,j} & 0 & 0 \\ 0 & w_{i,j}v_{i,1} & 0 & 0 & w_{i,j}v_{i,2} & 0 & 0 & w_{i,j}v_{i,3} & 0 & 0 & w_{i,j} & 0 \\ 0 & 0 & w_{i,j}v_{i,1} & 0 & 0 & w_{i,j}v_{i,2} & 0 & 0 & w_{i,j}v_{i,3} & 0 & 0 & w_{i,j} \end{bmatrix} \tag{2.4}$$

Multi-weight enveloping

Z této matice je vidět, že celému propojení je přidělena pouze jedna váha $w_{i,j}$. Metoda multi-weight enveloping tuto jednu váhu nahrazuje dvanácti, protože zobecněný tvar linear blend skinningu dovoluje dvanáct různých hodnot vah. Zároveň to však přináší problém, že tyto váhy je třeba nastavit při tvorbě modelu. Jejich návrh je také složitý, protože nemají tak jednoduchou a intuitivní interpretaci, jako váha linear blend skinningu. [21]

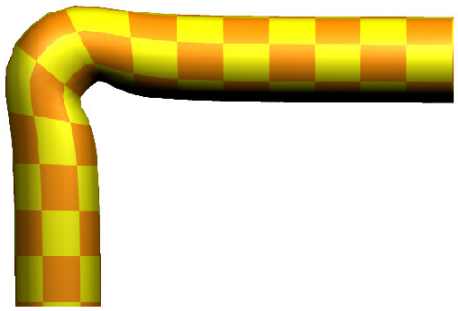
Animation space

Metoda animation space upravuje techniku multi-weight enveloping. Vychází ze zjištění, že všech dvanáct vah původní metody není užitečných a snižuje jejich počet na čtyři. Mohlo by se zdát, že ubrání vah využitelnost metody sníží, ale původní multi-weight enveloping dovoluje nastavení vah tak, že není zachována rotační invariance vůči souřadnicím světa. Pokud není rotační invariance zachována je možné, aby pouhá rotace modelu způsobila jeho deformaci, což je téměř vždy nevyžádaný efekt. [15]

2.4.3 Nelineární metody

Lineární metody mají efektivní implementaci a relativně snadno pochopitelnou matematiku. Pro zbavení candy-wrapper efektu však vyžadují více parametrů, kterými proces zesložiťují a i tak může docházet k nějakému nevyžádanému smršťování. Problém je v tom, že míchání trojrozměrných rotací což jsou prvky grupy SO(3), nerespektuje skutečnost, že grupa SO(3) je zakřivená varieta. Nelineární metody používají místo lineárního míchání takové, které zohledňuje vlastnosti variety.

Obrázek 2.11: Obrázky vlevo odpovídají linear blend skinningu a ty napravo dual quaternion skinningu. Z porovnání je vidět, že linear blend skinning v určitých pózách vykazuje problémy, které dual quaternion skinning řeší.[6]



Obrázek 2.12: Bulging artefakt, který vzniká při použití techniky dual quaternion skinning. [4]

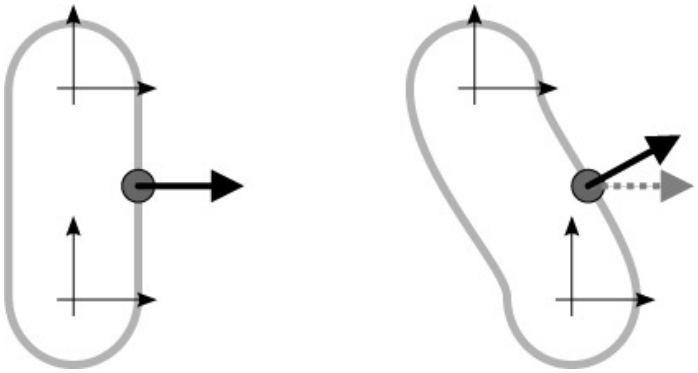
Dual quaternion skinning

Dual quaternion skinning je pokročilá technika používající kombinaci quaternionů a duálních čísel. Duální představují rozšíření reálných čísel o duální část. K reálným číslům přidávají ε, který má vlastnost, že $\varepsilon^2 = 0$.

Technika dual quaternion skinning používá kombinaci dvou kvaternionů (jeden pro rotaci a druhý pro posunutí). Rotace je představována reálným kvaternionem a posunutí duálním. Společně je lze namapovat do prostoru grupy SE(3) odpovídající rigidním transformacím. Z tohoto mapování vyplývá první z nedostatků metody a to, že duální kvaternion dokáže reprezentovat pouze rigidní transformace. Většina kosterních animací však nevyžaduje jiné než rigidní transformace, takže je pro ně technika použitelná.

Při mapování do prostoru grupy SE(3) ve skutečnosti dochází ke dvojitému překrytí, protože $\hat{q}$ a $-\hat{q}$ reprezentují stejnou tranformaci. To však nepředstavuje problém a naopak zlepšuje vlastnosti míchání transformací, protože to lépe předchází vzniku artefaktů jako je candy-wrapper. [6]

Dual quaternion skinning oproti multi-lineárním metodám nevyžaduje složitější nastavení vah a je plně zaměnitelný s technikou linear blend skinning. To představuje velikou výhodu, protože to dovoluje relativně jednoduché nahrazení skinningu v existujících řeše-



Obrázek 2.13: Na obrázku je znázorněn jednoduchý mesh transformovaný pomocí dvou kostí. Obrázek vlevo ukazuje výchozí pózu. Na obrázku vpravo je dolní kost přesunuta, ale není s ní otočeno. Černá šipka ukazuje skutečnou normálu a šedá šipka na obrázku vpravo normálu vypočítanou transformací pomocí metod jako je linear blend skinning nebo dual quaternion skinning. [17]

ních a nevyžaduje žádný zásah do existujících modelů. Tato technika sice řeší problémy kolapsu kloubů, které v určitých případech vykazuje linear blend skinning (porovnání na obrázku 2.11), neobejde se však bez vlastních problémů. Artefaktu, který se u ní vyskytuje, se říká bulging artefakt a způsobuje „napuchnutí“ kloubu a je ilustrován na obrázku 2.12.

2.4.4 Transformace normálových vektorů

Skinning se zabývá transformací vrcholů modelů, často je však vyžadováno i osvětlení animovaného modelu a pro to je potřeba transformovat i normály. Pro tento problém se nabízí několik řešení.

Je možné transformovat vrcholy modelu a normály z nich přesně vypočítat. Výpočty normál by tak však mohly začít až teprve poté, co jsou spočítány nové pozice vrcholů a vykreslování by vyžadovalo dvě fáze výpočtů. Kvůli paralelní struktuře grafických karet tohle není příliš vhodné řešení.

Další možností je využít stejnou transformaci, jaká je využita pro vrcholy. To se již běžně používá, ale výsledek obvykle obsahuje nepřesnosti. Obrázek 2.13 ukazuje důvod proč.

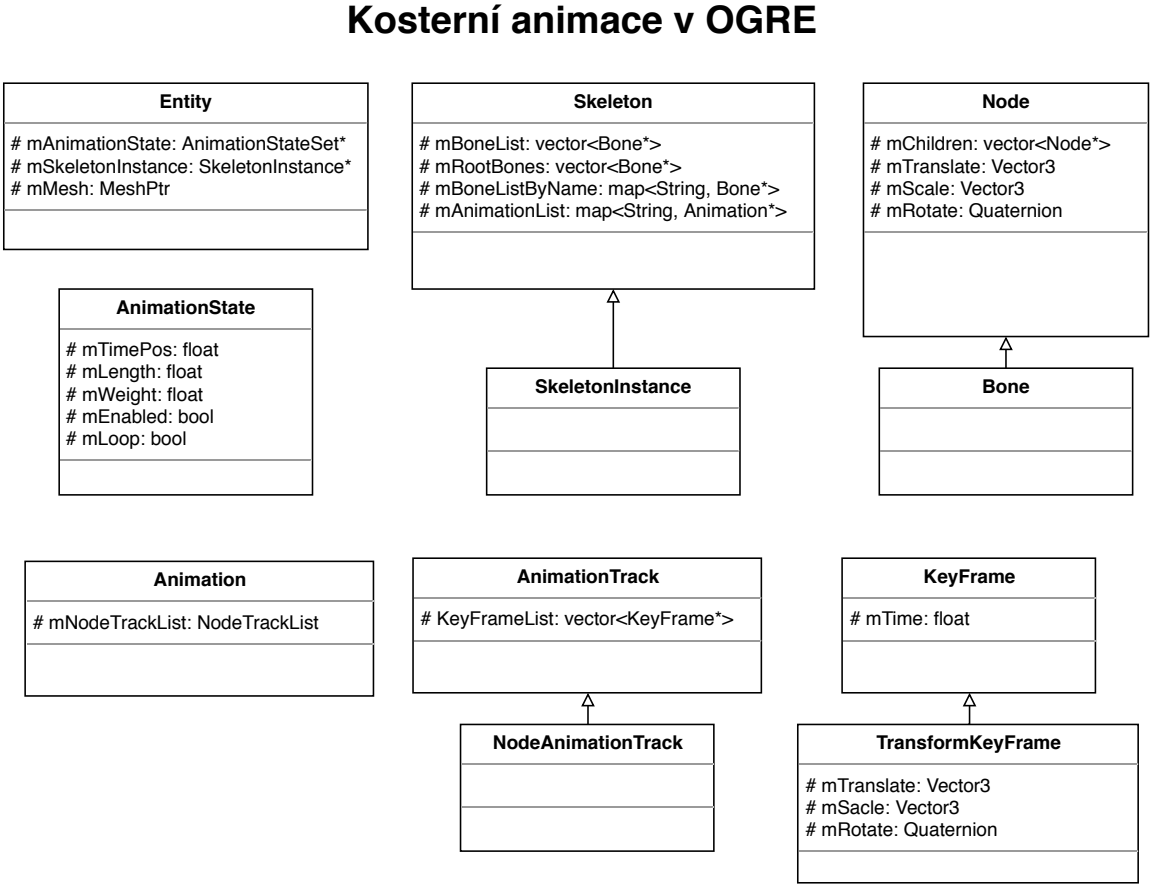
Pro správný výpočet normál při použití metody linear blend skinning je možné využít následující rovnici

$$\mathbf{n} = \overline{\left(\sum w_i M_i + M_i \mathbf{v}' \frac{\partial w_i}{\partial \mathbf{v}'}\right)}^{-T} \mathbf{n}' \tag{2.5}$$

, kde $\mathbf{n}'$ je normála v bind póze, $\mathbf{v}'$ je vrchol v bind póze. Toto řešení je o 5 % pomalejší než výše zmíněná aproximace a výsledné normálové vektory jsou spočítány přesně. [18]

2.5 Existující řešení systémů kosterních animací

Aby bylo možné vytvořit co možná nejlepší návrh, je vhodné prozkoumat existující řešení a inspirovat se jejich principy. V této části je prozkoumán 3D renderovací engine OGRE a toolkit OpenSceneGraph z hlediska implementace kosterních animací. Zvoleny byly z důvodu otevřeného zdrojového kódu a zaměření na vykreslování v reálném čase.



Obrázek 2.14: Diagram tříd OGRE souvisejících s kosterními animacemi.

2.5.1 OGRE

OGRE (Object-Oriented Graphics Rendering Engine) je open source 3D renderovací engine psaný v C++. Podporuje řadu funkcí a způsobů vykreslování včetně kosterních animací.

Na obrázku 2.14 jsou zobrazeny třídy relevantní pro kosterní animace. V OGRE je základem každého vykreslovaného objektu třída `Entity`, která slouží jako jeho instance. Obsahuje ukazatel na objekt třídy `Mesh`, což je grafická reprezentace modelu, která dále obsahuje objekty třídy `SubMesh`, reprezentující jednotlivá vykreslovací volání. Objekt třídy `Entity` nemusí obsahovat kostru, v tom případě se vykresluje běžným způsobem. Pro využití kosterních animací však kostru mít musí.

`Entity` se pomocí ukazatele odkazuje na třídu `SkeletonInstance`, která obsahuje její vlastní kopii kostry. `SkeletonInstance` dědí ze třídy `Skeleton`, která obsahuje pole kostí. Kosti jsou reprezentovány třídou `Bone` tvořící hierarchii transformací, díky podědění ze třídy `Node`. Nastavení těchto transformací vede na vytvoření nějaké pózy modelu.

Kostra dále obsahuje seznam animací díky ukazatelům na objekty třídy `Animation`. Animace je tvořena seznamem objektů třídy `NodeAnimationTrack`, kde každý z nich ovládá transformaci jednoho objektu `Node` (v případě kosterních animací `Bone`). Výsledná kosterní animace vzniká tak díky kombinaci animací několika či všech kostí. Třída `AnimationTrack` drží seznam klíčových snímků s transformacemi pro daný uzel (kost).

Poslední částí kosterního systému OGRE je třída `AnimationState`, která určuje stav jedné animace entity. Entita jich může obsahovat více a to dovoluje míchání animací s nastavenými různými vahami.

2.5.2 OpenSceneGraph

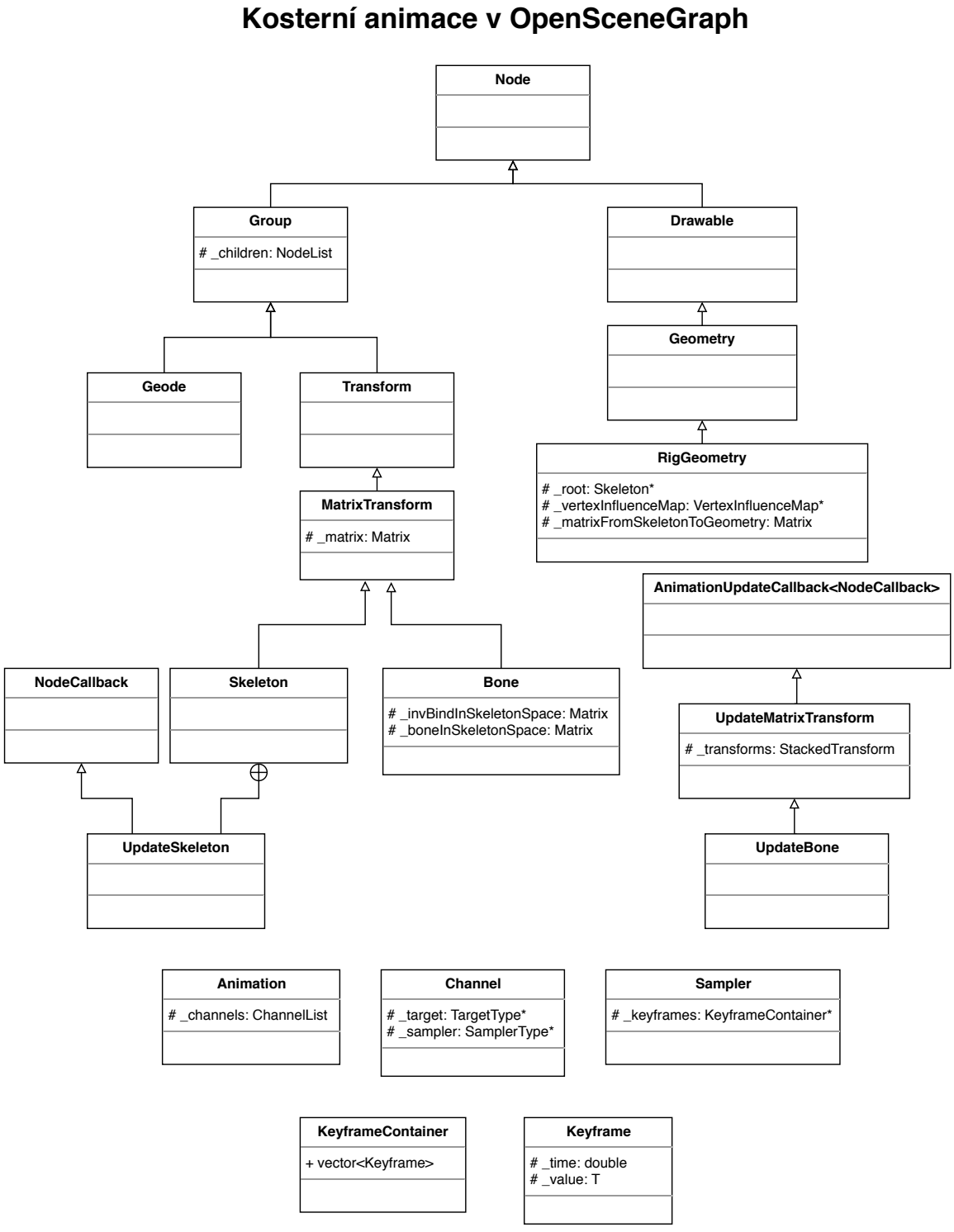
OpenSceneGraph je open source toolkit pro 3D grafiku založený na grafu scény. Graf scény je strom tvořící rozhraní mezi aplikací a nízkoúrovňovým vykreslovacím API. Jsou v něm uloženy všechny potřebné informace pro vykreslování i animaci. Aplikace ho obvykle vytvoří a následně volá různé typy jeho průchodů. Pro kosterní animace jsou důležité update traversal a draw traversal, které způsobují aktualizaci a vykreslení grafu.

Na obrázku 2.15 jsou třídy OpenSceneGraph relevantní pro kosterní animace. Základem grafu scény je třída `Node`. Všechny objekty přímo ležící v grafu z ní musí dědit. Další důležitou třídou je `Group`, která umožňuje uzlu držet potomky a tím pádem tvořit strom. Vykreslitelné objekty musí dědit ze třídy `Drawable` a jsou umisťovány do uzlů `Geode`, který může obsahovat pouze `Drawable` potomky. `Geometry` představuje geometrii vykreslitelnou jedním vykreslovacím voláním. `Transform` uzly slouží pro transformaci hierarchie potomků ve scéně například pomocí matic jako v případě `MatrixTransform`.

Animace je reprezentována třídou `Animation`, která obsahuje seznam tříd `Channel`. `Channel` se zaměřuje na jeden cíl (např.: matici trasformačního uzlu) a čerpá data z objektu třídy `Sampler`. Ten vzorkuje animační data dle svého nastavení, může také provádět jejich interpolaci. `Sampler` se odkazuje na klíčové snímky přes `KeyframeContainer`.

Dosud popsané třídy slouží pro vykreslování a animaci modelů běžným způsobem (bez použití kostry). Pro využití kostry je třeba přidat objekt třídy `Skeleton` někam do grafu scény a hierarchii objektů `Bone` jako jejo potomky. `Drawable` objekty umístěné do kostí lze již animovat pomocí kostí, pro použití skinningu je však třeba využít třídu `RigGeometry`. Ta ke geometrii přidává informace o ovlivnění vrcholů kostmi a umístí se jako potomek třídy `Skeleton`.

Callback `UpdateBone` se stará o aktualizaci kosti při průchodu grafem scény. Uloží se do něho výchozí pojmenované animované hodnoty ve formě tříd `StackedTransform` a jeho jméno musí být nastaveno na stejné jako je jméno kosti. Dále je třeba ho nastavit jako update callback kosti. `Skeleton` obsahuje výchozí update callback jako vnořenou třídu a při jeho zavolání aktualizuje své potomky, pokud je nutná validace kostry. Jména kanálů je třeba nastavit na stejné jméno jako bylo nastaveno u příslušné animované hodnoty a jméno cíle na jméno příslušné kosti. Kanály ovládající jednotlivé transformace jednotlivých kostí jsou přidány do třídy `Animation`, která již může kosterní animaci přehrávat. [20]



Obrázek 2.15: Diagram tříd OpenSceneGraph souvisejících s kosterními animacemi (třídy na dolní části diagramu týkající se animací byly zjednodušeny, kvůli použití templátů, princip je zachován).

Kapitola 3

Návrh rozšíření

Cílem této práce je přidat systém pro použití kosterních animací do knihovny GPUEngine. Tato kapitola tedy nejprve popisuje GPUEngine a především jeho součásti využitelné pro rozšíření. Následuje analýza znovupoužití vhodných částí GPUEnginu a možnosti zakomponování systému do knihovny. Poté jsou definovány požadavky a obecný návrh systému. Nakonec je popsán princip využití datových struktur knihovny Assimp, která bude použita pro načítání modelů ze souborů.

3.1 GPUEngine

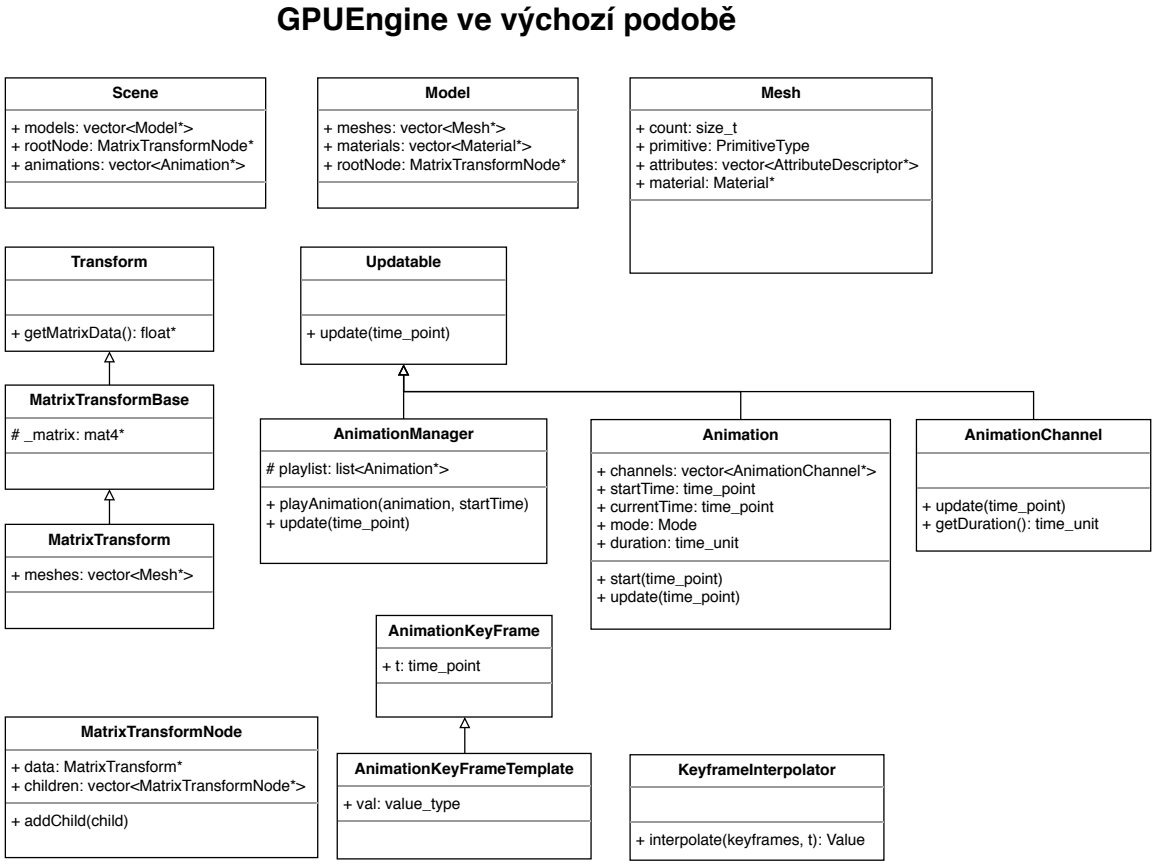
GPUEngine je knihovna vyvíjená na Fakultě informačních technologií VUT v Brně. Je určena pro pomoc s vykreslováním při vývoji 3D aplikací. Je multiplatformní, napsána v C++ a využívá konfiguračního systému CMake. Pro vykreslování podporuje OpenGL, ale jeho použití nevyžaduje a volbu konkrétních technologií a dalších knihoven (např.: pro práci s okny) nechává na uživateli. Následující popis se týká větve `srcInclude_merge` GPUEnginu, ze které tato práce vychází [1].

GPUEngine se skládá z několika částí. `geCore` obsahuje základní definice. `geGL` zjednodušuje práci s OpenGL a poskytuje jeho objektové zapouzdření. `geSG` obsahuje graf scény, animace, prostředky pro testování vzájemné polohy objektů a další pomocné třídy nezávislé na použitém vykreslovacím rozhraní. `geUtil` sdružuje další pomocné třídy a funkce například pro práci s kamerou či načítání souborů. `geAd` není přímou součástí knihovny, ale obsahuje třídy pro spolupráci s jinými knihovnami. Pro tuto práci je nejdůležitější `geSG`, protože obsahuje animace a graf scény, a díky tomu je to vhodné místo pro přidání systému kosterních animací.

Na obrázku 3.1 je diagram tříd GPUEnginu, které se týkají vykreslování a animací. Graf scény je reprezentován třídou `Scene`. Ta obsahuje pole modelů (třídy `Model`) a model obsahuje pole meshů (třídy `Mesh`). `Mesh` reprezentuje jedno vykreslovací volání. Je definován počtem vrcholů, typem geometrického primitiva, seznamem atributů pro shader a materiálem, který udává barvu (texturu). `Model` dále obsahuje seznam všech materiálů svých meshů a ukazatel na kořenový uzel hierarchie transformací s meshi, která ho definuje.

Druhá položka třídy `Scene` je kořenový uzel, obsahující hierarchii grafu scény. Základem těchto uzlů je transformace. Abstraktní třída `Transform` představuje obecnou transformac vyžadující pouze funkci na získání ukazatele na data v maticové formě. Z ní dědí třída `MatrixTransformBase` představuje transformaci interně využívající matici o čtyřech

[1] https://github.com/Rendering-FIT/GPUEngine/tree/srcInclude_merge



Obrázek 3.1: Diagram tříd původní verze GPUEnginu souvisejících s vykreslováním pomocí grafu scény a animacemi.

řádcích a sloupcích. `MatrixTransform` do transformace přidává pole ukazatelů na meshe. `MatrixTransformNode` nakonec dělá z této transformace uzel, který může obsahovat potomky a sám může být potomek jiného uzlu.

Jako poslední je ve `Scene` pole animací. Základem animací GPUEnginu je abstraktní třída `AnimationChannel`. Obvykle se používá pro aktualizování jedné hodnoty či skupiny souvisejících hodnot (např.: pozice, orientace, barva). Více kanalů pak může být sjednoceno ve třídě `Animation`, která provádí synchronní spouštění a aktualizaci všech svých kanálů. Přidává také možnost spouštění v režimu smyčky. `AnimationManager` nakonec umožňuje sdružené přehrávání více animací. Všechny tyto animační třídy používají rozhraní abstraktní třídy `Updatable`.

Poslední pro tuto práci relevantní třídy jsou pouze pomocné (mohou pomoct s animacemi, ale není explicitně vyžadováno jejich použití). `AnimationKeyFrame` je třída obsahující časové razítko umožňující nalezení hodnot podle času pro snadnou interpolaci. `AnimationKeyFrameTemplate` je šablonová třída přidávající hodnotu a jejím poděděním lze již vytvářet klíčové snímky s potřebnými hodnotami. `KeyframeInterpolator` spolupracuje s těmito klíčovými snímky dokáže je interpolovat. Součástí GPUEnginu jsou interpolátory umožňující nejbližší, lineární a sférickou lineární interpolaci.

3.2 Analýza

Ve vytvářeném rozšíření je vhodné co nejvíce využít existující možnosti knihovny a nově implementovat pouze nezbytné části. GPUEngine již podporuje ukládání a vykreslování otexturovaných modelů. Model deformovaný kosterními animacemi lze vykreslit stejným způsobem (odlišnost je v transformacích prováděných v shaderech). Dále je potřeba zachovat kompatibilitu grafu scény tak, aby podporoval původní i nové (kosterně animované) modely. Knihovna již také obsahuje transformace a možnost vytvořit jejich hierarchii, což může být užitečné pro tvorbu kostry.

Způsob, jakým třída `Animation` sdružuje animační kanály, je podobný jako v knihovnách OGRE a OpenSceneGraph popsaných v části 2.5. V nich je při použití kosterních animací pro každou kost kostry použit jeden kanál. Třída animace obsahuje tyto kanály ovládající kosti a představuje kosterní animaci. Této podobnosti je vhodné využít a použít animace a animační kanály stejným způsobem.

Hodnoty tříd klíčových snímků lze libovolně zvolit, a tak je lze bez problému využít pro uložení transformací kostí. Stejně tak lze využít interpolátory pro jejich interpolaci.

Mezi pomocnými třídami GPUEnginu v `geAd` se také nachází `AssimpModelLoader`. Tato třída využívá knihovnu Assimp pro načítání modelů. Jelikož načítání klasických modelů je v ní již implementováno a zároveň Assimp podporuje kosterní animace, je příhodné využít stávající implementace a pouze ji obohatit o načítání kostry a kosterních animací. Způsob uložení dat knihovnou Assimp je popsán v sekci 3.4.

3.3 Návrh a požadavky

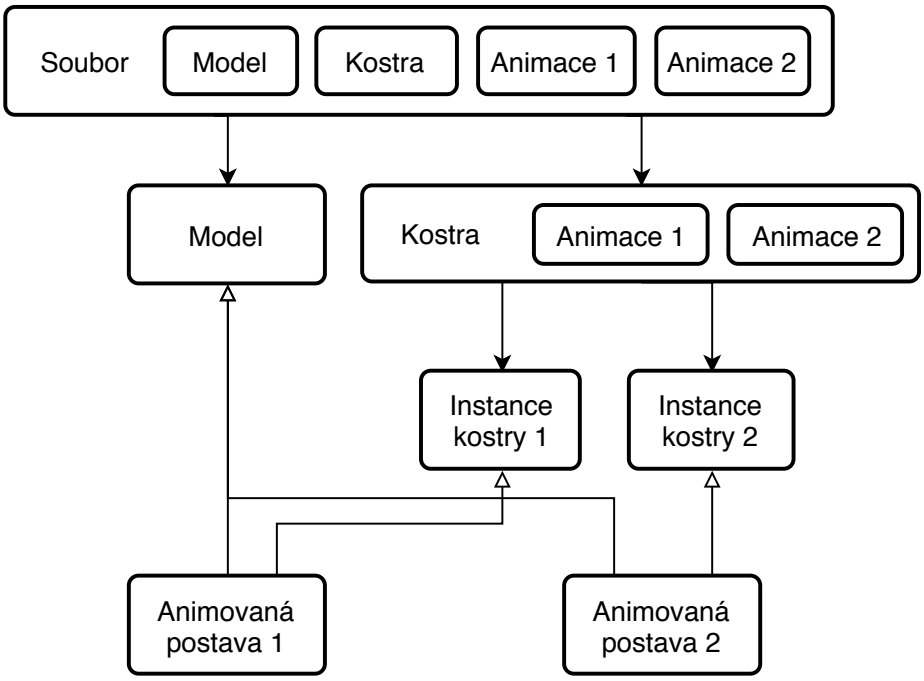
Dobrý systém kosterních animací by měl podporovat animování více stejných modelů najednou nezávisle na sobě. Zároveň by se však měl snažit o efektivní využití paměti, což vyžaduje co nejmenší míru redundance. Je tedy dobré zajistit, aby model a animační data nebyla duplikována. Obrázek 3.2 ukazuje návrh jakým způsobem toho bude docíleno. Model, kostra a animace jsou v návrhu načteny a uloženy v paměti pouze jednou. Pro každý vykreslovaný model animovaný podle kostry je vytvořena její vlastní instance. Oddělenost instancí koster dovoluje nastavení libovolné pózy jedné postavy, aniž by to ovlivnilo druhou. Instance koster však lze animovat podle stejných animačních dat.

Další aspekt kosterních animací, který jsem se rozhodl v rozšíření podporovat, je míchání animací pomocí vah. Díky tomu bude možné plynule přecházet z jedné animační sekvence do druhé.

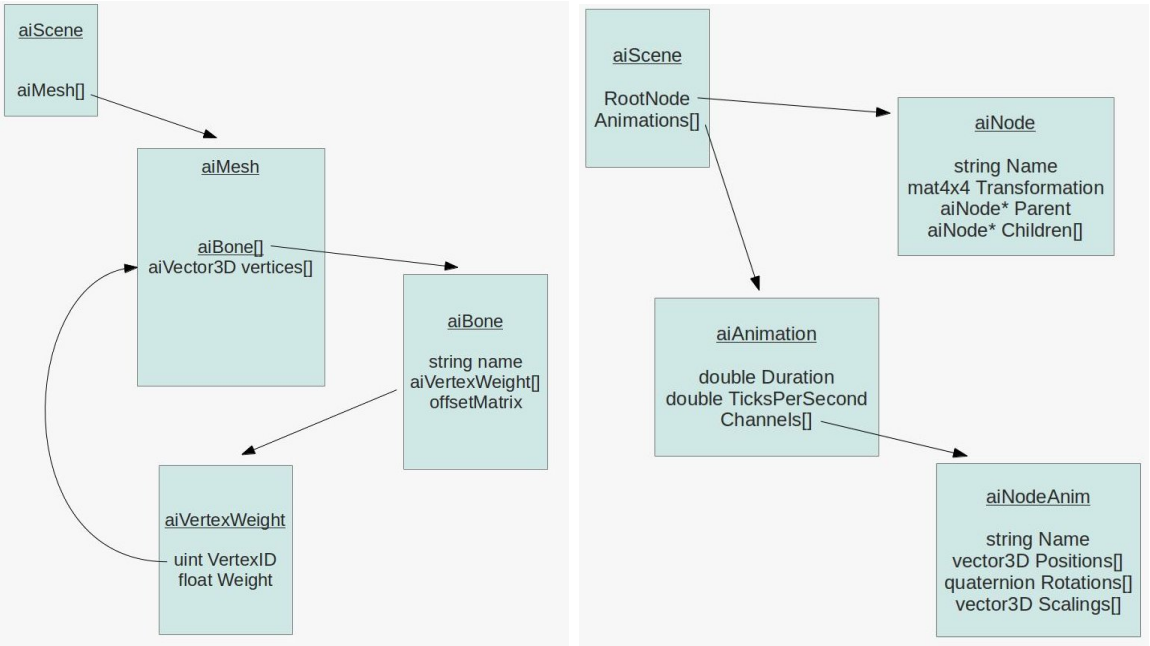
3.4 Načtení modelu s animacemi ze souboru

Existuje spousta formátů pro ukládání trojrozměrných modelů. Pro tuto práci je vyžadován takový z nich, který podporuje i kosterní animace. GPUEngine již obsahuje podporu pro základní načítání modelů pomocí knihovny Assimp. Jelikož knihovna Assimp podporuje kosterní animace a umožňuje načítání z řady používaných formátů uniformním způsobem, je vhodné jí využít[9]. Tato část popisuje obvyklý způsob zpracování tohoto uniformního formátu, do kterého Assimp načítá soubory.

Strukturu dat po načtení modelu Assimpem lze vidět na obrázku 3.3 vlevo. Výchozím bodem je aiScene představující kořen celého modelu. Uvnitř aiScene se nachází pole struktur aiMesh, což je část modelu pokrytá jednou texturou. Mesh poté obsahuje pole svých kostí



Obrázek 3.2: Ve výsledném systému je třeba zajistit co nejmenší redundanci dat v paměti. Model a animační data se v ní nachází pouze jednou. Jednotlivé instance animovaných modelů mají vlastní instanci kostry a tak je možné přehrávat jejich animace nezávisle na sobě.



Obrázek 3.3: Struktura dat načtených pomocí knihovny Assimp. Vlevo jsou data důležitá pro získání kostí. Vpravo se nachází data potřebná pro načtení animací a hierarchie kostí. [14]

a vrcholů. Každá kost obsahuje své jméno a váhy s indexy do pole vrcholů. Právě tyto vrcholy kost ovlivňuje. Dále její offsetová matice určuje transformaci potřebnou pro přesun vrcholu z prostoru meshe do lokálního prostoru kosti. [8]

Renderování však obyčejně probíhá po vrcholech, ne po kostech. Z toho důvodu je potřeba data projít a přeuspořádat tak, aby každý vrchol nesl informace o kostech, které ho ovlivňují. Při vykreslování pak pole kosterních transformací přijde do uniformní proměnné a vrcholy se podle nich mají možnost transformovat.

Informace o kostech jsou tímto získány, ale ještě nic nevíme o jejich hierarchii. K tomu slouží další struktury obsažené v aiScene. Na obrázku 3.3 vpravo je znázorněna struktura dat potřebných pro získání hierarchie kostí a animací. V aiScene se nachází kořenový uzel hierarchie struktur aiNode, které tvoří strom. Struktura aiNode obsahuje ukazatel na rodičovský uzel, pole ukazatelů na podřízené uzly a transformaci ze svého prostoru do prostoru rodiče. Dále může být volitelně pojmenovaná. Pokud má představovat kost, toto jméno musí být shodné se jménem kosti, které odpovídá. Kostru tedy získáme tak, že podle jmen nalezneme v hierarchii aiNode všechny kosti ze struktury aiMesh, a poté umístění v této hierarchii odpovídá umístění v kostře.

Poslední částí je pole struktur aiAnimation, které je také umístěno v aiScene. Každá tato struktura reprezentuje posloupnost snímků animace jako je chůze, běh, střelba atd. Interpolací mezi snímky dojde k odpovídající animaci. Doba animace je zde určena v počtu animačních kroků (Duration) a zároveň je zde počet kroků za sekundu (TicksPerSecond). Pole kanálů potom obsahuje pojmenované aiNodeAnim pro každou animovanou kost, a ta musí mít v hierarchii uzlů stejné jméno.

V aiNodeAnim jsou umístěny tři pole položek obsahující pozici, rotaci a scale. V každé z nich je animační čas a transformace. K získání výsledné transformace v daném čase je

třeba nalézt dvě hodnoty s nejbližším časem a interpolovat mezi nimi. Transformace se poté zkombinují do jedné, a tím vznikne obvykle matice. Pro výpočet konečné transformace kosti se jde od kořenu hierarchie ke kosti a pokaždé je získána výsledná transformace v požadovaném čase. Tyto transformace jsou řetězeny, až se dojde do dané kosti. [14]

Kapitola 4

Implementace

Implementace proběhla v C++, což vyplývá z jazyka, ve kterém je knihovna napsána. Je využito konfiguračního systému CMake[1], který knihovna také používá.

Nejprve je popsána implementace systému kosterních animací a výsledná verze GPUEnginu, tak jak byl do knihovny začleněn. Následuje popis rozšíření třídy `AssimpModelLoader` o načítání kostry a kosterních animací do nových struktur GPUEnginu. Dále je obecně popsán způsob jakým systém použít v aplikaci. Nakonec je popsána vytvořená aplikace, která toto použití demonstruje.

4.1 Systém kosterních animací

Pro výslednou implementaci systému bylo třeba uvažovat všechny potřeby a omezení Assimpu, GPUEnginu a návrhu, které byly zmíněny v kapitole 3. Tato sekce nejprve vysvětluje řešení několika z těchto omezení a problémů. Následuje konkrétní popis celého animačního systému.

4.1.1 Řešené implementační problémy

Jedním z řešených implementačních problémů je, že každá načtená transformace v rámci modelu nemusí být kostí (transformuje část modelu, ale není animována). Tento problém lze vyřešit několika způsoby. Lze ji považovat za kost, ale v tom případě je potřeba její transformaci spolu s ostatními transformacemi kostí posílat v uniformní proměnné do shaderu. Toto řešení zabírá zbytečné místo v uniformní paměti shaderů a tím pádem tak omezuje maximální počet kostí. Kdyby počet takových transformací převažoval počet kostí, byl by tento způsob velice neefektivní. Já jsem zvolil druhou možnost řešení, která tkví v oddělení transformací a kostí. Přidává sice několik tříd navíc, ale do shaderů je posíláno jenom to, co je tam skutečně potřeba.

Dalším nelehkým úkolem bylo zajistit způsob jak uskutečnit míchání různých kosterních animací. Výsledné řešení to sice neumožňuje, systém je však připraven pro budoucí podporu. Animace a animační kanály GPUEnginu jsou založeny na rozhraní třídy `Updatable`, jak lze vidět na obrázku 3.1. Tato třída vyžaduje schopnost aktualizace pomocí volání metody s jediným parametrem časové hodnoty. Podle Liskov Substitution Principle [13] je třeba, aby se odvozené třídy daly použít na místě bázových. Nelze tedy přidat další parametry (metody), obsahující váhu míchání animací a zároveň dodržet principy správného objektového návrhu.

[1] https://cmake.org/

Zároveň by nebylo vhodné vytvářet nové animační třídy, když už v GPUEnginu existují. Tento problém byl vyřešen tím, že transformační uzly vykreslovaného modelu neobsahují pouze jednu transformaci, ale jejich frontu. Fronta může být zaplněna transformacemi různých animací a před vykreslením dojde k jejich smíchání.

4.1.2 Detaily Implementace

Na obrázku 4.1 lze vidět diagram obsahující relevantní třídy GPUEnginu a nově implementované třídy kosterního rozšíření. Implementované rozšíření je dostupné v repozitáři na stránce gitlab.com[2].

Společná část modelu

Pro reprezentaci modelu je využita původní třída `Model`. Třída `RigModel` ji propojuje s třídou `Skeleton`, která představuje kostru a obsahuje animace. `RigModel` tím představuje kompletně načtený model ze souboru.

Kostra (`Skeleton`) obsahuje pole kostí. Kosti jsou pojmenovány a obsahují transformaci z prostoru modelu do prostoru kosti (využívá se pro skinning). Dále drží ukazatel na původní transformaci modelu, která dané kosti přísluší. To je z důvodu provázání transformací a kostí.

Animace jsou také součástí kostry. Jedná se o pole, ve kterém každá položka představuje jednu kosterní animaci. U animací bylo třeba vyřešit problém s jejich mícháním tak jak je popsáno v předchozí podkapitole. Protože `RigModel` ještě neobsahuje fronty určené k míchání animací, je zde využito tříd `SkeletalAnimationSetChannel` (uložených v `Animation`), které jsou zaměřeny na jednu konkrétní matici v hierarchii modelu.

Instance modelu

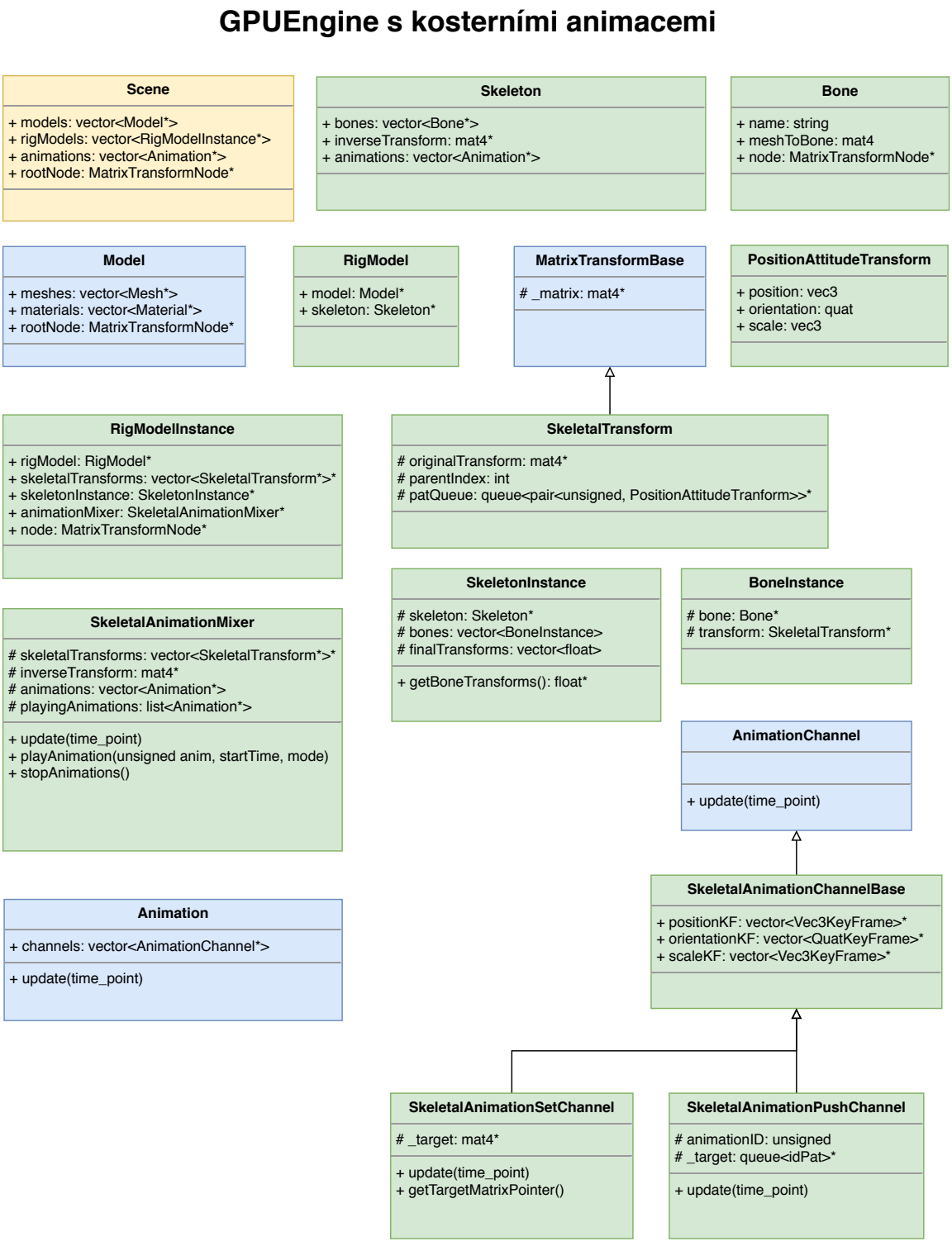
`RigModel` je společný pro všechny `RigModelInstance` představující instance vykreslovaných kosterních modelů, ty na něho proto odkazují. Dále obsahují ukazatel na pole ukazatelů tříd `SkeletalTransform`. Jsou zde použity dvě úrovně ukazatelů. Celé pole musí být totiž přístupné ze třídy `SkeletalAnimationMixer`, která provádí míchání animací. Zároveň však je potřeba, aby se na jednotlivé transformace mohli odkazovat instance kostí.

Třída `SkeletalTransform` dědí z `MatrixTransformBase` matici, která udává výslednou transformaci. Dále obsahuje ukazatel na matici původní transformace a frontu. Ukazatel slouží pro případ, že nedojde ke změně transformace a bude třeba použít tu původní. Do fronty jsou ukládány jednotlivé transformace z animací určené ke smíchání (tento proces bude přesněji specifikován níže). Nakonec obsahuje index rodičovské `SkeletalTransform`, který se využije při výsledném výpočtu transformací. Poloha kosti totiž závisí na poloze nadřazené kosti, jak je popsáno v teoretické části 2.3.

Další položkou `RigModelInstance` je `SkeletonInstance` obsahující pole instancí kostí. Instance kosti propojuje třídy `Bone` a `SkeletalTransform`. Ze `SkeletalTransform` instance získá výslednou transformaci a `Bone` je třeba pro transformaci z prostoru modelu do prostoru kosti, která je pro správné zobrazení modelu nutná. Každá instance kosti odpovídá transformaci posílané do shaderu (blíže bude popsáno v části 4.3), ke kterému je vyhrazeno pole `finalTransforms`.

`SkeletalAnimationMixer` se stará o animace instance modelu. Obsahuje ukazatel na jeho transformace, aby v nich mohl provádět míchání animací. Dále vlastní všechny ani-

[2] https://gitlab.com/xminar30/gpuengineskeletalanimations



Obrázek 4.1: Diagram všech tříd implementovaného rozšíření. Modrou barvou jsou vyznačeny původní třídy GPUEnginu, žlutou modifikované a zelenou nově implementované.

mace instance a má informaci o právě přehrávaných. Jeho animace se liší od animací třídy `Skeleton`. Sdílejí sice animační data (a nedochází tak k redundanci jak je popsáno v 3.3), ale používají odlišné animační kanály. `SkeletalAnimationPushChannel` zná číslo své animace a cíl nad kterým pracuje není obyčejná matice, ale fronta transformací, kterou obsahuje třída `SkeletalTransform`. Tyto transformace jsou uloženy zvlášť jako pozice, orientace a scale (ve formě třídy `PositionAttitudeTransform`). To je z důvodu interpolace mezi nimi, která je po složkách robustnější. Stejným způsobem jsou také uloženy klíčové snímky všech animací, takže nepřibývají výpočty navíc. Uložení klíčových snímků tímto způsobem má ještě jednu výhodu. Dovoluje to rozdíl v počtu klíčových snímků jednotlivých složek. Scale se v kosterních animacích často nepoužívá, a tak animace může obsahovat pouze jednu jeho hodnotu pro všechny časy.

Umístění instance modelu do grafu scény

Aby nebylo zasaženo do stávající struktury třídy `Scene`. Umístění do grafu scény je řešeno ukazatelem nacházejícím se v `RigModelInstance`. Graf scény může obsahovat meshe běžným způsobem. Poloha instance modelu je transformována podle transformace uzlu, na který ukazuje.

4.2 Načtení modelu s animacemi ze souboru

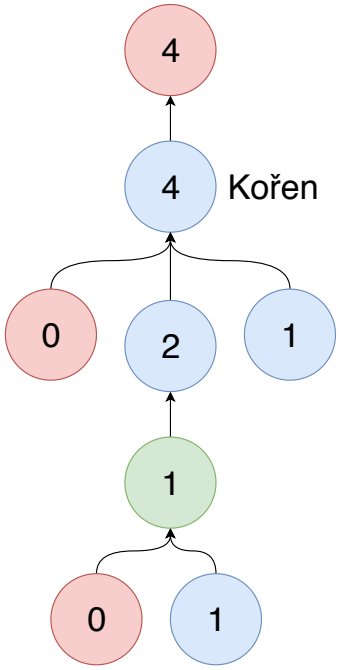
Jak již bylo zmíněno v části 3.2, k načítání modelů s kosterními animacemi je využito knihovny Assimp. Datové struktury této knihovny jsou již popsány v 3.4. Tato podkapitola se zabývá rozšířením třídy `AssimpModelLoader`, kteoru již GPUEngine obsahuje, o načítání kosterních animací.

Složitější formáty pro ukládání 3D modelů jsou schopny uložit celé rozsáhlé scény s desítkami postav a dalších entit. Zpracování takových scén by však bylo složité. Většina formátů a 3D modelářů používá soubory takovým způsobem, že jeden soubor odpovídá jednomu 3D modelu. Z toho důvodu omezení, že načítaný soubor bude brán jako jeden kosterní model, je přijatelné.

`AssimpModelLoader` bere jako parametr název souboru a vrací objekt třídy `Scene` z obrázku 4.1. Rozhodnutí, zda se jedná o obyčejný či kosterní model, se provádí podle přítomnosti kostí v meshích načteného modelu. Materiály a meshe jsou v obou případech zpracovány stejným způsobem (tato část již byla implementována v GPUEnginu).

Dále je popsán způsob zpracování předpokládající existenci kostí. Do atributů vrcholů jsou přidány informace o indexech a vahách kostí určené pro skinning. Limit je nastaven tak, že jeden vrchol může být ovlivňován maximálně čtyřmi kostmi. Použití více kostí obvykle nepřináší téměř žádné vizuální zlepšení. Propojení kostí s hierarchií transformací je provedeno pomocí `std::map`, kde se mapuje název a ukazatel na kost. Tohoto principu je dále využito několikrát pro další propojení (animací s transformacemi, při vytváření `RigModelInstance` z `RigModel`).

Následuje zpracování hierarchie kostí. Aby nedocházelo k plýtvání prostředky, je zajištěno, že načtená hierarchie obsahuje pouze nezbytné uzly. Toho je docíleno přiřazením čísla všem uzlům (inicializovaným na nulu). Následným zjištěním polohy všech kostí (animovaných transformací). A pro každou z nich inkrementací čísla uzlu všem uzlům po cestě ke kořenu. Uzly s nulou je možné vynechat, protože nemají na model vliv. Stejně tak lze vynechat všechny transformace směrem od kořene, jejichž děti obsahují stejné číslo jako



Obrázek 4.2: Modrá barva reprezentuje kosti. Zelená reprezentuje transformace, které jsou nezbytné ve výsledné kostře. Červená reprezentuje transformace, které se vynechají.

ony. Kořenem hierarchie modelu se stává první potřebná transformace. Tento princip je ilustrován na obrázku 4.2.

Po zpracování kostí jsou načteny animace a tím je vytvořen celý objekt třídy `RigModel`. Z něho je poté vytvořen objekt třídy `RigModelInstance`. Ten je nakonec umístěn do nového uzlu scény, kterou `AssimpModelLoader` vrací.

4.3 Použití v aplikaci

Pro použití kosterního systému v aplikaci je třeba mít nainstalovaný GPUEngine, všechny jeho závislosti a k aplikaci přidat `AssimpModelLoader`. Pomocí něho je pak možné načíst kosterní model s animacemi ze souboru do scény. Scéna poté obsahuje všechny potřebné informace o tom co vykreslit, konkrétní způsob vykreslení již závisí na aplikaci (stejně jako obvykle při použití GPUEnginu).

Scéna může být vykreslena libovolným grafickým API. Tento popis bude používat konvence OpenGL, princip by měl však být podobný a aplikovatelný i pro ostatní API.

Meshe nacházející se v modelech grafu scény je třeba zpracovat a nahrát do paměti grafické karty. V rámci knihovny GPUEngine lze použít `geGL` pro ulehčení práce s OpenGL. Každý mesh musí minimálně obsahovat atribut s vrcholy. Dále může obsahovat například indexy, normály a texturovací souřadnice. V případě meshe animovaného pomocí kosterních animací obsahuje tento mesh další dva atributy a to indexy kostí a váhy kostí. Ty jsou poté použity ve vertex shaderu pro skinning modelu. V případě, že není vyžadován skinning nemusejí být meshe vůbec použity. Například v případě vykreslování pouze kostry je vyžadována pouze hierarchie transformací modelu.

Obrázek 4.3: Snímek z demonstrační aplikace ukazující postavy animované pomocí kosterních animací a vykreslené pomocí skinningu (je použita základní technika linear blend skinning popsaná v části 2.4.1). Na obrázku je vidět, že postavy jsou navzájem v odlišné póze a tudíž odlišné fázi animace.

Aby vykreslované meshe mohly být deformovány podle kostí, je třeba ve vertex shaderu znát jejich transformace. K tomu lze použít například uniformní proměnnou. Třída `SkeletonInstance` obsahuje funkci `getBoneTransforms()`. Ta vrací ukazatel na pole floatů, ve kterém jsou v maticové podobě uloženy transformace všech kostí modelu. Toto pole lze jedním příkazem nahrát do uniformní proměnné shaderu a obvykle se tak dělá při každém snímku, aby byla animace plynulá.

Shadery je možné použít podle potřeby, programátor si je však musí vytvořit sám. Pro vykreslení kostry modelu stačí jednoduchý shader pro vykreslení meshe představujícího kost. Poté ho vykreslovat pro každou transformaci hierarchie, kde se matice modelu nastaví právě na tuto transformaci. Pro základní vykreslení kosterního modelu je možné použít jednoduchý linear blend skinning shader (technika popsána v části 2.4.1). Pro složitější skinning může být použita například technika dual quaternion skinning (zmíněno v sekci 2.4.3).

4.4 Demonstrační aplikace

Demonstrační aplikace je založena na příkladu `Simple_QtgeSG`, který je součástí GPUEnginu. Tento příklad ukazuje načtení a vykreslení jednoduchého modelu s texturami, a proto byl vhodný pro úpravu a demonstraci modelu s kosterními animacemi. Pro testování byl použit 3D model ve formátu MD5, který obsahuje jednu animační sekvenci[3]. Vytvořená aplikace demonstruje načtení modelu, kosterní animace a vytvoření druhé nezávislé instance

[3]Model je volně stažitelný z https://www.katsbits.com/download/models/md5-example.php

modelu. Dále je ukázána technika jednoduchého skinningu. Na obrázku 4.3 lze vidět snímek z aplikace. Výsledná implementace je dostupná v git repozitáři na stránce gitlab.com[4].

Pro konfiguraci a podporu různých operačních systémů využívá aplikace systému CMake. Pro práci s oknem a vytvoření OpenGL kontextu je využito knihovny Qt[5]. Dále je využito dvou pomocných tříd GPUEnginu z jeho části `geAd`. `AssimpModelLoader` je použit pro načtení modelu s kosterními animacemi. `QtImageLoader` slouží pro načtení textur s pomocí již zmíněné knihovny Qt.

V aplikaci je model načten do scény. Následně je vytvořen nový posunutý uzel scény, do kterého je umístěna druhá instance modelu. Animace instancí jsou spuštěny s rozdílem jedné sekundy, aby bylo vidět, že každá instance má ve stejném čase jinak transformované kosti.

Jelikož umístění instance modelu ve scéně provedeno pomocí ukazatele z instance na uzel, před vykreslováním je pro urychlení vytvořeno mapování z uzlu na instanci. Při vykreslování je procházen graf scény, spojovány transformace v jeho uzlech a pokud je uzel mapován na instanci kosterního modelu, instance je v tom místě vykreslena.

V shaderech aplikace je ke skinningu navíc implementováno jednoduché difuzní stínování. Pro které bylo třeba pomocí kostí transformovat nejen vrcholy, ale i normály modelu.

[4] https://gitlab.com/xminar30/skeletalanimationexample
[5] https://www.qt.io/

Kapitola 5

Možnosti budoucí práce

V této práci byl implementován základ kosterního systému, který už je použitelný pro jednoduché hry a další 3D aplikace. Existuje však řada dalších vylepšení a technik, kterými je možné ho zdokonalit a rozšířit.

První z věcí, které se nabízejí, je implementace míchání různých animací (princip popsán v podkapitole 2.3.3). Návrh s ním již počítá a potřebné struktury jsou na místě. Pro jeho uskutečnění je třeba implementovat míchání transformací ve třídě `SkeletalAnimationMixer` a nalézt vhodný model s více animačními sekvencemi, na kterém by to šlo otestovat.

Dalším možným rozšířením je vytvoření částí shaderového kódu, které by byly připojovány k shaderům aplikace. Použití skinningu by pak bylo zajištěno například voláním nějaké vyhrazené vestavěné funkce v shaderu. Takových částí shaderového kódu by mohlo být několik, pro různé způsoby zobrazení kostry a techniky skinningu.

Další z budoucích prací by se mohla zabývat rozšířením kosterních animací o animace obličeje například pomocí techniky morph targets popsané v části 2.1.3.

Pokročilejší práce by se mohly zabývat inverzní kinematikou, jejíž základní princip je popsán v podkapitole 2.3.5. To by mohlo umožňovat tvorbu nových animací přímo v aplikaci, doposud se kosterní systém spoléhá pouze na předem vytvořená a načtená data. V rámci inverzní kinematiky lze také implementovat omezení na jednotlivé klouby kostry. Například u ramenního kloubu modelu člověka existuje pouze část uvěřitelných úhlů jeho rotace. Systém s daným omezením by nedovolil nastavené hranice překročit.

Nabízí se také propojení fyzikální simulace s kosterními animacemi pro tvorbu procedurálních animací. Například rag doll animací popsaných v části 2.3.6.

Kapitola 6

Závěr

Tato práce se zabývala tvorbou rozšíření pro podporu kosterních animací do knihovny GPUEngine.

V rámci práce byly nastudovány techniky animací, kosterních animací a skinningu. Dále byly prozkoumány kosterní systémy knihoven OGRE a OpenSceneGraph.

V části návrhu byla nastudována existující část knihovny GPUEngine a analyzován způsob jejího rozšíření. Následně byl navrhnut způsob, jak nezvyšovat redundanci dat v paměti při použití více instancí stejných modelů.

Podle návrhu bylo úspěšně implementováno základní kosterní rozšíření do knihovny GPUEngine. Jeho použití bylo demonstrováno jednoduchou aplikací.

Při práci se ukázalo, že ač je základní princip kosterních animací jednoduchý, navrhnout rozšiřitelný systém, který by počítal s budoucí podporou řady možností kosterních animací je složité. Výsledné řešení je přímo rozšiřitelné minimálně o míchání animací, což je metoda, které se například při vývoji videoher dá dobře využít. Není vyloučené, že pro některá rozšíření zmiňovaná v předchozí kapitole bude třeba nějaké části systému změnit.

Literatura

[1] Buss, S. R.: Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. Technická zpráva, IEEE Journal of Robotics and Automation, 2004.

[2] Erleben, K.; Andrews, S.: Inverse Kinematics Problems with Exact Hessian Matrices. In *Proceedings of the Tenth International Conference on Motion in Games*, MIG '17, New York, NY, USA: ACM, 2017, ISBN 978-1-4503-5541-4, s. 14:1–14:6, doi:10.1145/3136457.3136464.
URL http://doi.acm.org/10.1145/3136457.3136464

[3] Gregory, J.: *Game Engine Architecture, Second Edition*. Natick, MA, USA: A. K. Peters, Ltd., druhé vydání, 2014, ISBN 1466560010, 9781466560017.

[4] Jacobson, A.; Deng, Z.; Kavan, L.; aj.: Skinning: Real-time Shape Deformation. In *ACM SIGGRAPH 2014 Courses*, 2014.

[5] Kavan, L.; Collins, S.; O'Sullivan, C.: Automatic Linearization of Nonlinear Skinning. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, I3D '09, New York, NY, USA: ACM, 2009, ISBN 978-1-60558-429-4, s. 49–56, doi:10.1145/1507149.1507157.
URL http://doi.acm.org/10.1145/1507149.1507157

[6] Kavan, L.; Collins, S.; Žára, J.; aj.: Geometric Skinning with Approximate Dual Quaternion Blending. *ACM Trans. Graph.*, ročník 27, č. 4, Listopad 2008: s. 105:1–105:23, ISSN 0730-0301, doi:10.1145/1409625.1409627.
URL http://doi.acm.org/10.1145/1409625.1409627

[7] Kochanek, D. H. U.; Bartels, R. H.: Interpolating Splines with Local Tension, Continuity, and Bias Control. *SIGGRAPH Comput. Graph.*, ročník 18, č. 3, Leden 1984: s. 33–41, ISSN 0097-8930, doi:10.1145/964965.808575.
URL http://doi.acm.org/10.1145/964965.808575

[8] Kulling, K.: Assimp: Data Structures.
URL http://sir-kimmi.de/assimp/lib_html/data.html

[9] Kulling, K.: Open Asset Import Library. [Online; navštíveno 14.01.2019].
URL http://www.assimp.org/

[10] Lewis, J. P.; Cordner, M.; Fong, N.: Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-driven Deformation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH

’00, New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, ISBN 1-58113-208-5, s. 165–172, doi:10.1145/344779.344862.
URL http://dx.doi.org/10.1145/344779.344862

[11] Magnenat-Thalmann, N.; Laperrière, R.; Thalmann, D.: Joint-dependent Local Deformations for Hand Animation and Object Grasping. In *Proceedings on Graphics Interface ’88*, Toronto, Ont., Canada, Canada: Canadian Information Processing Society, 1988, s. 26–33.
URL http://dl.acm.org/citation.cfm?id=102313.102317

[12] Magnenat-Thalmann, N.; Thalmann, D.: The Direction of Synthetic Actors in the Film Rendez-Vous a Montreal. *IEEE Computer Graphics and Applications*, ročník 7, č. 12, Dec 1987: s. 9–19, ISSN 0272-1716, doi:10.1109/MCG.1987.276934.

[13] Martin, R. C.: The Principles of OOD.
URL http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod

[14] Meiri, E.: Skeletal Animation With Assimp.
URL http://ogldev.atspace.co.uk/www/tutorial38/tutorial38.html

[15] Merry, B.; Marais, P.; Gain, J.: Animation space: A truly linear framework for character animation. *ACM Trans. Graph*, ročník 25, 2006: str. 2006.

[16] Mukundan, R.: Skeletal Animation. In *Advanced Methods in Computer Graphics: With examples in OpenGL*, London: Springer London, 2012 vydání, 2012, ISBN 9781447123392, s. 53–76.

[17] Tarini, M.; Panozzo, D.; Sorkine-Hornung, O.: Accurate and Efficient Lighting for Skinned Models. *Comput. Graph. Forum*, ročník 33, č. 2, Květen 2014: s. 421–428, ISSN 0167-7055, doi:10.1111/cgf.12330.
URL http://dx.doi.org/10.1111/cgf.12330

[18] Tarini, M.; Panozzo, D.; Sorkine-Hornung, O.: Accurate and Efficient Lighting for Skinned Models. *Comput. Graph. Forum*, ročník 33, č. 2, Květen 2014: s. 421–428, ISSN 0167-7055, doi:10.1111/cgf.12330.
URL http://dx.doi.org/10.1111/cgf.12330

[19] Thalmann, N. M.: *Interactive computer animation*. London: Prentice-Hall, 1996, ISBN 0-13-518309-X.

[20] Wang, R.: *OpenSceneGraph 3 cookbook : over 80 recipes to show advanced 3D programming techniques with the OpenSceneGraph API*. Birmingham, UK: Packt Pub./Open Source, 2012, ISBN 978-1-84951-688-4.

[21] Wang, X. C.; Phillips, C.: Multi-weight Enveloping: Least-squares Approximation Techniques for Skin Animation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA ’02, New York, NY, USA: ACM, 2002, ISBN 1-58113-573-4, s. 129–138, doi:10.1145/545261.545283.
URL http://doi.acm.org/10.1145/545261.545283